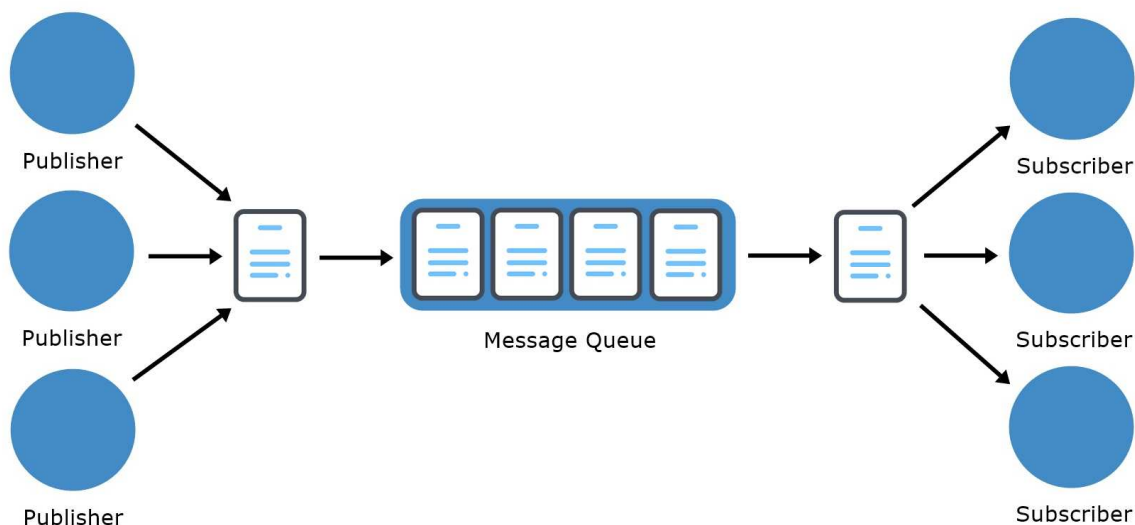


Rad sa redovima poruka

Red poruka je oblik asinhronne komunikacije koji se koristi u bezserverskim i mikroservisnim arhitekturama. Poruke se čuvaju u redu dok se ne obrade i izbrišu. Svaka poruka koja dospe u red obrađuje se samo jednom, od strane jednog korisnika. U modernim cloud arhitekturama, aplikacije su razdvojene u manje nezavisne gradivne blokove koje je lakše razviti, primeniti i održavati. Redovi poruka pružaju komunikaciju i koordinaciju za ove distribuirane aplikacije. Oni mogu značajno pojednostaviti kodiranje nevezanih aplikacija, istovremeno poboljšavajući performanse, pouzdanost i skalabilnost.

Red poruka sadrži bafer, koji privremeno skladišti poruke, i krajnje tačke, koje omogućavaju softverskim komponentama da se povežu sa redom radi slanja i primanja poruka. Poruke su obično male i mogu biti stvari poput zahteva, odgovora, poruka o greškama ili samo obične informacije. Da bi poslala poruku, komponenta koja se zove **publisher** dodaje poruku u red. Poruka se čuva u redu dok druga komponenta, **subscriber**, ne preuzme poruku i ne uradi nešto sa njom.

U softverskoj arhitekturi, **publish-subscribe** je obrazac za razmenu poruka kod kojeg pošiljaoci poruka ne programiraju poruke tako da se direktno šalju određenim primaocima, već objavljene poruke kategorizuju u klase ne znajući koji pretplatnici ih traže, ako oni uopšte postoje. Slično tome, primaoci izražavaju interesovanje za jednu ili više klasa i primaju samo poruke koje ih zanimaju, ne znajući od kojih pošiljalaca dolaze, ako ih ima.



Slika 8.1. Prikaz publish-subscribe obrasca na primeru reda poruka

Redovi u MongoDB-u

Radi boljeg razumevanja funkcionisanja redova, najpre ćemo prikazati jedan jednostavan primer koristeći MongoDB. Redovi u MongoDB-u funkcionišu isto kao i redovi iz realnog sveta, po FIFO principu. Drugim rečima, elementi koji pristižu dodaju se na kraj reda, dok se elementi koji se uklanjaju iz reda uzimaju sa njegovog početka.

MongoDB podržava i više redova. U nastavku lekcije biće prikazano kako je moguće skladištiti redove u MongoDB u vidu kolekcije čiji su elementi parovi ključ–vrednost. Za početak je neophodno da modul pymongo uvedemo u naš program kako bismo mogli da koristimo sve MongoDB funkcije, što činimo komandom `import pymongo`.

Kada smo uveli ovaj mogul u program, prvi neophodan korak je povezivanje klijenta na server pozivanjem `pymongo.MongoClient()` metode. Ova metoda će kao parametre primiti server i port kao instrukcije za povezivanje, što će u našem slučaju biti lokalni server i podrazumevani MongoDB port. Ovako kreiranog klijenta čuvamo u promenljivoj i povezujemo ga na bazu podataka `test` kao što je to prikazano u primeru ispod.

Primer povezivanja klijenta na bazu podataka u MongoDB-u:

```
import pymongo

c = pymongo.MongoClient('localhost', 27017)
db = c['test']
```

Zatim ćemo u našu tabelu `test`, koja predstavlja kolekciju baze podataka, umetnuti dva dokumenta, red `q1` i red `q2`. Umetanje elemenata u red vrši se pomoću funkcije `insert_one()` u željenom redu, kojoj se prosleđuje element reda u vidu para ključ–vrednost. U nastavku je prikazano umetanje elemenata u dva reda.

Primer umetanja elemenata u redove u MongoDB-u:

```
db.q1.insert_one({'val': 'first message'})
db.q1.insert_one({'val': 'second message'})
db.q1.insert_one({'val': 'third message'})

db.q2.insert_one({'val': 'queue2 first message'})
db.q2.insert_one({'val': 'queue2 second message'})

for row in db.q1.find():
    print (row)
for row in db.q2.find():
    print (row)
```

Naša kolekcija podataka sada ima dva reda podataka, red `q1` i red `q2`. Pozivanjem metode `find()` u okviru redova moguće je izvršiti pregled svih elemenata koji se u tom redu, nalaze kao što je to prikazano na slici 8.2.

Elements of q1

```
{ "_id" : ObjectId("51d2baaa3402f8b2dec93888"), "val" : "first message" }  
{ "_id" : ObjectId("51d2bab73402f8b2dec93889"), "val" : "second message" }  
{ "_id" : ObjectId("51d2babe3402f8b2dec9388a"), "val" : "third message" }
```

Elements of q2

```
{ "_id" : ObjectId("51d2bafd3402f8b2dec9388b"), "val" : "queue2 first message" }  
{ "_id" : ObjectId("51d2bb013402f8b2dec9388c"), "val" : "queue2 second message" }
```

Slika 8.2. Prikaz elemenata redova q1 i q2 korišćenjem funkcije find()

Uklanjanje elemenata iz reda vrši se na nešto drugačiji način. Metoda koja se u ovu svrhu koristi je `find_one_and_delete()`, koja prima parametar, koji predstavlja ključ-vrednost podatak. Ključ predstavlja ključ iz kolekcije a vrednost je potrebno da postavimo da predstavlja vrednost koju želimo da izbrišemo.

Primer uklanjanja podataka iz redova u MongoDB-u:

```
asd = db.q1.find_one_and_delete({'val': 'first_message'})  
  
print(asd['val'])
```

Metoda za uklanjanje elemenata iz reda izvršavanjem gore definisane komande ukloniće umetnuti element za koji definišemo ključ i vrednost. Kako bismo se uverili u to za *q1* kolekciju baze podataka, ispisaćemo sve elemente koji su skladišteni, kao što je prikazano u nastavku.

Primer prikazivanja reda podataka u MongoDB-u:

```
for row in db.q1.find():  
    print(row)
```

Elements of q1

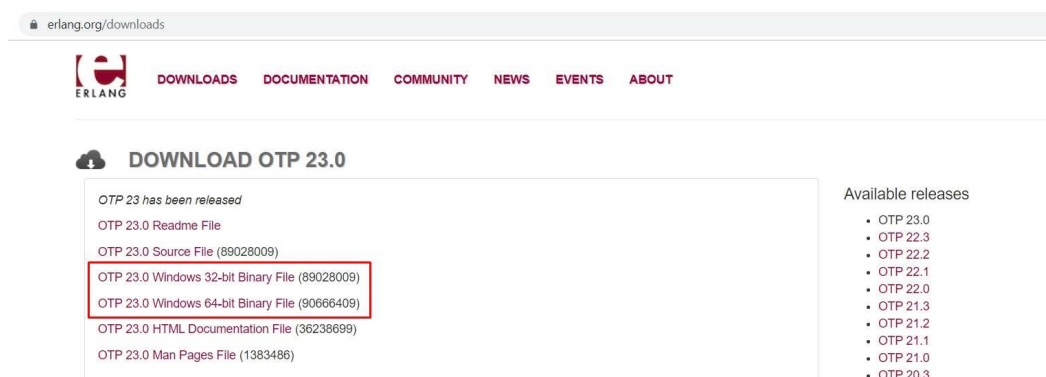
```
{ "_id" : ObjectId("51d2bab73402f8b2dec93889"), "val" : "second message" }  
{ "_id" : ObjectId("51d2babe3402f8b2dec9388a"), "val" : "third message" }
```

Slika 8.3. Elementi q1 reda nakon uklanjanja prvog elementa

Izmeniti kod tako da pravite 2 niza pomoću `pymongo` modula koji sadrže parne odn. neparne brojeve od 1 do 10. Ispisati oba reda.

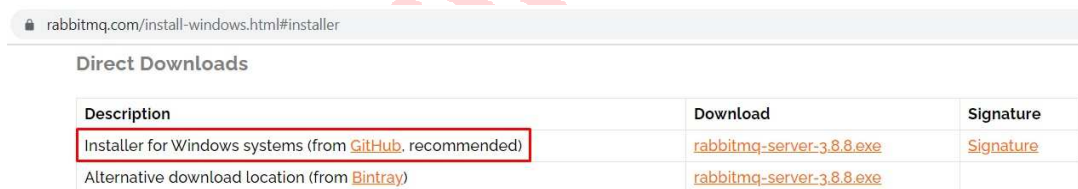
Redovi u RabbitMQ-u

RabbitMQ predstavlja softver za razmenu poruka otvorenog koda. S obzirom na to da je napisan na funkcionalnom jeziku Erlang, za njegovo korišćenje je neophodna instalacija ovog jezika, koju je moguće pronaći na [zvaničnom sajtu](#). Na samom početku ove stranice nalaze se različite verzije instalacije (slika 8.4). U našem primeru odabraćemo *Windows Binary File* verziju koja odgovara našoj arhitekturi računara. Nakon preuzimanja fajla, njegovim pokretanjem izvršićemo instalaciju.



Slika 8.4. Zvanična stranica za preuzimanje instalacije Erlang funkcionalnog jezika

Sledeći neophodan korak je instalacija RabbitMQ softvera, koja se može pronaći na ovom [zvaničnom linku](#). U okviru stranice ćemo pronaći direktna preuzimanja instalacija i odabrati preporučenu verziju koju je postavio GitHub (slika 8.5). Nakon preuzimanja ovog fajla izvršićemo instalaciju njegovim pokretanjem.



Slika 8.5. Direktna preuzimanja instalacije u okviru zvanične stranice RabbitMQ softvera

Treća neophodna stavka za funkcionisanje ovog softvera jeste klijent, kojeg obezbeđujemo instalacijom **pika** modula zadavanjem naredbe `python -m pip install pika --upgrade` u okviru komandne linije računara.

Napomena

Postoji veliki broj klijenata za RabbitMQ na mnogo različitih jezika. U našem primeru koristićemo Pika 1.0.0 modul, koji je Python klijent koji preporučuje RabbitMQ tim.

Za kreiranje konekcije koristimo pika modul, koji prethodno uvodimo u program pomoću `import pika` linije koda. U nastavku je prikazano kako se pomoću ovog modula može napraviti konekcija na lokalni server.

Primer konekcije na server korišćenjem pika modula u RabbitMQ-u:

```
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
```

Metoda `pika.BlockingConnection()` stvara sloj na vrhu pika asinhronog jezgra, pružajući metode koje će blokirati dok se ne vrati njihov očekivani odgovor. Za konekciju na bazu koristi se `pika.ConnectionParameters()`, koji za parametar prima server na koji će se izvršiti konekcija. Ukoliko bismo želeli da se povežemo na drugu mašinu, umesto `localhost` stavili bismo IP adresu te mašine. Metoda `connection.channel()` pruža omotač za interakciju sa RabbitMQ-om implementirajući metode i attribute.

Sledeći korak je kreiranje reda u koji će se poruke slati. Ukoliko ne postoji red primaoca, RabbitMQ će ispustiti poruku. Red se kreira pomoću `queue_declare()` metode, u okviru koje se deklariše naziv reda u vidu parametra. Kreiranje ćemo izvršiti u okviru `sender.py` programa.

Primer deklarisanja reda za slanje poruka u RabbitMQ:

```
channel.queue_declare(queue='hello')
```

Sada kada imamo definisan red, slanje poruke možemo izvršiti pomoću `basic_publish()` metode. Ova metoda kao parametar prima `exchange`, što predstavlja način razmene poruka; `routing_key`, odnosno red u koji želimo da pošaljemo poruku, i njen sadržaj – `body`.

Primer slanja poruke u red *hello* korišćenjem RabbitMQ-a:

```
channel.basic_publish(exchange='',
                      routing_key='hello',
                      body='Hello World!')

print(" [x] Sent 'Hello World!'")
```

U navedenom primeru poslata je poruka *Hello World!* u deklarisanu red *hello*. Parametar `exchange` u ovom primeru je prazan string, jer je reč o direktnom slanju poruke u red. Međutim, u praksi se poruke u red nikada neće slati direktno.

Uzmimo kao primer način razmene **fanout**, koji za ulogu ima emitovanje poruka na sve poznate redove. Način razmene moguće je definisati pomoću `exchange_declare()` metode. Kada se deklariše naziv ove razmene, ona se kasnije preko tog naziva može pozvati u okviru metode slanja. U tom slučaju, red u koji se poruke šalju ne može se specifično definisati.

Primer slanja poruke u sve poznate redove korišćenjem RabbitMQ-a:

```
channel.exchange_declare(exchange='msgs',
                        exchange_type='fanout')
```

```
channel.basic_publish(exchange='msgs',
                      routing_key='',
                      body='Message')
```

Po završetku slanja poruka preporučljivo je isprazniti bafer i zatvoriti konekciju, kao što je to učinjeno dalje u primeru.

Primer zatvaranja konekcije u RabbitMQ-u:

```
connection.close()
```

Pitanje

Koja od sledećih metoda se koristi za definisanje pošiljaoca poruka?

- `basic_publish()`
- `basic_consume()`
- `basic_send()`
- `basic_receive()`

Objašnjenje:

Za definisanje pošiljaoca poruka koristi se `basic_publish()` metoda.

Naš drugi program, **receiver.py**, primaće poruke iz reda i ispisivati ih na ekranu. Za primanje poruka neophodno je povezivanje na server identično kao i kod slanja. Da bismo se osigurali da red postoji, opet ćemo ga deklarirati. Bez obzira na to koliko puta izdamo naredbu za kreiranje reda, samo će jedan biti kreiran.

Primer deklarisanja reda za prijem poruka u RabbitMQ-u:

```
channel.queue_declare(queue='hello')
```

Primanje poruka iz reda je nešto složenije i vrši se pomoću `callback()` metode. Kad god red primi poruku, pika biblioteka poziva ovu funkciju. U našem slučaju, ova funkcija će na ekranu ispisati sadržaj poruke.

Primer deklarisanja reda za prijem poruka u RabbitMQ-u:

```
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

channel.basic_consume(queue='hello',
                      auto_ack=True,
                      on_message_callback=callback)
```

Metoda `basic_consume()` definiše na koji način će poruke iz reda biti preuzimane. Ona kao parametar prima red – queue iz kojeg će podaci biti preuzeti. Parametar `auto_ack` koji je podešen na vrednost `True` označava automatsko potvrđivanje uspešno primljenih poruka u redu. Parametar `on_message_callback` treba da odredi funkciju koja će biti izvršena po prijemu poruke, a to u našem slučaju predstavlja ispis koji je definisan pomoću `callback` metode.

Na kraju, definišemo jednu neprekidnu petlju koja čeka podatke i pokreće `callback()` funkciju kad god podatak pristigne u red. Za slučaju prekida programa, definisana je i funkcija `KeyboardInterrupt()`.

Primer deklarisanja beskonačne petlje za prijem podataka u red u RabbitMQ-u:

```
print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

Pogledajmo u nastavku kako sada izgledaju naši programi **sender.py** i **receiver.py**.

Primer sender.py programa za slanje poruka u red u RabbitMQ-u:

```
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))

channel = connection.channel()

channel.queue_declare(queue='hello')

channel.basic_publish(exchange='',
                     routing_key='hello',
                     body='Hello World!')

print(" [x] Sent 'Hello World!'")

connection.close()
```

Primer receiver.py programa za prijem poruka u red u RabbitMQ-u:

```
import pika, sys, os

def main():
```

```

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))

channel = connection.channel()

channel.queue_declare(queue='hello')

def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

channel.basic_consume(queue='hello',
                      on_message_callback=callback,
                      auto_ack=True)

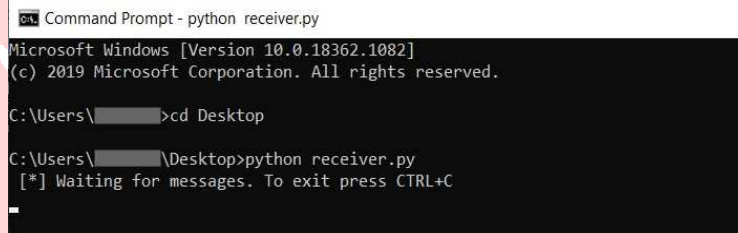
print(' [*] Waiting for messages. To exit press CTRL+C')

channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

```

Kako bismo imali realnu sliku o tome kako ove poruke funkcionišu, u komandnoj liniji ćemo otvoriti program primaoca tako što ćemo se pozicionirati u direktorijum gde se taj fajl nalazi i zadati komandu `python receiver.py`. U komandnom prozoru dobićemo informaciju o čekanju na poruku kao na slici 8.6.



```

Command Prompt - python receiver.py
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>cd Desktop

C:\Users\>\Desktop>python receiver.py
[*] Waiting for messages. To exit press CTRL+C

```

Slika 8.6. Pokretanje primaoca poruka receiver.py

Zatim ćemo pokrenuti program pošiljaoca kako bismo slali poruke u red. Na slici 8.7. prikazano je kako pošiljalac šalje tri poruke u red u okviru **sender.py** programa.


```
Command Prompt
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>cd Desktop

C:\Users\>\Desktop>python sender.py
[x] Sent 'Hello World!'

C:\Users\>\Desktop>python sender.py
[x] Sent 'Hello World!'

C:\Users\>\Desktop>python sender.py
[x] Sent 'Hello World!'
```

Slika 8.7. Pokretanje pošiljaoca poruka sender.py

Poslate poruke će sada izvršiti izmene na definisanom redu. Kao rezultat ova tri pokretanja sender.py programa, u komandnom prozoru primaoca prikazaće se informacije o primljenim porukama, što je moguće uočiti na slici 8.8.

```
Command Prompt - python receiver.py
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>cd Desktop

C:\Users\>\Desktop>python receiver.py
[*] Waiting for messages. To exit press CTRL+C
[x] Received b'Hello World!'
[x] Received b'Hello World!'
[x] Received b'Hello World!'
```

Slika 8.8. Prikaz programa primaoca nakon prijema poruka u hello redu

Rezime

- Red poruka je oblik asinhronne komunikacije koja se koristi u bezserverskim i mikroservisnim arhitekturama. Poruke se čuvaju u redu dok se ne obrade i izbrišu.
- Da bi poslala poruku, komponenta koja se zove *publisher* dodaje poruku u red. Poruka se čuva u redu dok druga komponenta, *subscriber*, ne preuzme poruku i ne uradi nešto sa njom.
- MongoDB podržava i više redova, koje skladišti u vidu kolekcije čiji su elementi parovi ključ-vrednost.
- Prvi neophodan korak za rad sa redovima je povezivanje klijenta na server pozivanjem `pymongo.MongoClient()` metode. Ova metoda će kao parametre primiti server i port kao instrukcije za povezivanje.
- Umetanje elemenata u red vrši se pomoću funkcije `insert_one()` u željenom redu, kojoj se prosleđuje element u vidu para ključ-vrednost.
- Metoda za uklanjanje elemenata je `find_and_modify()`. Ova metoda u vidu parametra `remove`, koji predstavlja ključ-vrednost podatak, koristi funkciju `pop` za izbacivanje poslednjeg elementa iz reda podešavanjem njegove vrednosti na -1.

- RabbitMQ je softver za razmenu poruka otvorenog koda napisan na funkcionalnom jeziku Erlang. Za njegovu upotrebu potrebna je instalacija jezika Erlang, programa RabbitMQ i modula pika.
- Za konekciju na bazu koristi se `pika.ConnectionParameters()`, koji za parametar prima server na koji će se izvršiti konekcija.
- Metoda `connection.channel()` pruža omotač za interakciju sa RabbitMQ-om implementirajući metode i ponašanja.
- Red se u RabbitMQ-u kreira pomoću metode `queue_declare()`, u okviru koje se deklariše naziv reda u vidu parametra.
- Pošiljalac poruka definiše se pomoću metode `basic_publish()`. Ova metoda kao parametar prima `exchange`, što predstavlja način razmene poruka, `routing_key`, odnosno red u koji želimo da pošaljemo poruku, i njen sadržaj – `body`.
- Način razmene moguće je definisati pomoću metode `exchange_declare()`. Ukoliko način razmene nije definisan, njegova podrazumevana vrednost je prazan string i poruka će biti poslata u red definisan parametrom `routing_key`.
- Kada je način razmene definisan, ciljani red, odnosno `routing_key` će biti prazan string.
- Primalac se definiše pomoću metode `basic_consume()`. U okviru primaoca, parametar `on_message_callback` treba da primi funkciju `callback()`, koja će biti izvršena po prijemu poruke. Tu funkciju je potrebno definisati.
- Na kraju je neophodno definisati jednu neprekidnu petlju koja čeka podatke i pokreće ranije definisanu `callback()` funkciju kad god podatak pristigne u red.

